Self-adaptive Synchronous Localization and Mapping using Runtime Feature Models

Christopher Werner, Sebastian Werner, René Schöne, Sebastian Götz and Uwe Aßmann
Software Technology Group, Technische Universität Dresden, Dresden, Germany

Keywords: Self-adaptive Systems, Feature Modeling, Robotics, Slam.

Abstract: Mobile autonomous robotic systems need to operate in unknown areas. For this, a plethora of simultaneous localization and mapping (SLAM) approaches has been proposed over the last decades. Although many of these existing approaches have been successfully applied even in real-world productive scenarios, they are typically designed for specific contexts (e.g., in- vs. outdoor, crowded vs. free areas, etc.). Thus, for different contexts, different SLAM algorithms should be used. In this paper, we propose a feature-based classification of SLAM algorithms and a reconfiguration approach to switch between existing SLAM implementations at runtime. By this, mobile robots are enabled to always use the most efficient implementation for their current contexts.

1 INTRODUCTION

Simultaneous localization and mapping (SLAM) is a standard problem in robotic software engineering, which covers the ability of a mobile robot to orient itself in an unknown environment. The robot has to perform two tasks simultaneously: it has to discover its environment (mapping) and has to estimate its own position in this environment (localization). For this, various sensors like ultrasonic, cameras, or LIDAR (light detection and ranging) are used.

Over the last decades, vast amounts of SLAM algorithms have been proposed in the literature, each of which can be considered as a specialized solution for a certain application context or certain operation conditions. For example, different algorithms have been proposed for in- and outdoor scenarios, for static and dynamic environments, for crowded and sparse areas, and for surface and underwater scenarios.

Typically, each new SLAM algorithm is developed from scratch without reusing parts of existing algorithms. In consequence, no common framework for SLAM algorithms, which allows to (de-)compose SLAM algorithms, exists yet. Moreover, as SLAM is used by mobile robots, the operation conditions of the robot can change at runtime. For example, by leaving the building, the operation conditions switch from in- to outdoor.

Thus, a mobile robot should be able to switch its SLAM algorithm at runtime whenever its operation conditions change. For this, a unifying framework, which explicitly covers the commonalities and differences between existing SLAM algorithms is required.

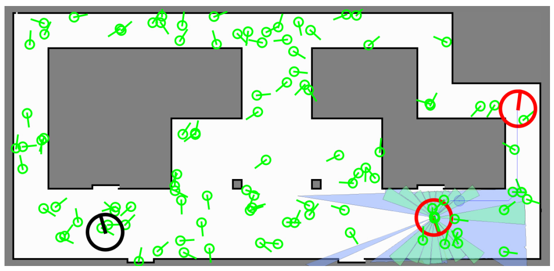
As a result, in this paper, we present such a unifying framework for SLAM algorithms represented as a feature model based on a commonality and variability analysis of over 30 existing implementations. Using this framework, we answer two research questions:
RQ1) Is it possible to systematically reuse parts of existing SLAM algorithms?
RQ2) Is it possible to dynamically exchange parts of existing SLAM algorithms during execution?

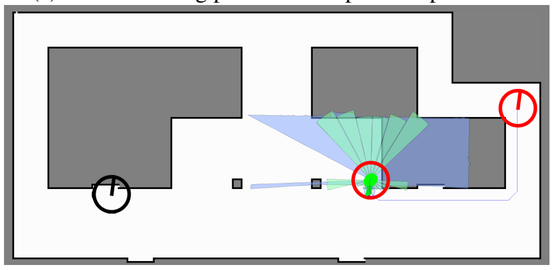
Additionally, we present a runtime reconfiguration approach, which allows switching between SLAM algorithms based on this feature model.

We show the general applicability of our approach by a case study implemented within the GeneralRobot framework (developed at HTW Dresden).

The remainder of this paper is structured as follows. In the next section, we introduce a running example, which will be used throughout the paper for illustrative purposes. Next, in Section 3, the unifying framework is discussed. The dynamic reconfiguration approach is captured in Section 4. The evaluation of our approach is presented in Section 5. We demarcate our approach from related work in Section 6. Finally, in Section 7, we conclude the paper and give pointers for future work.

Werner, C., Werner, S., Schöne, R., Götz, S. and Aßmann, U.
Self-adaptive Synchronous Localization and Mapping using Runtime Feature Models.
DOI: 10.5220/0006945504090418
In *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)*, pages 409-418
ISBN: 978-989-758-318-6


(a) KLD in wrong position with possible positions.


(b) KLD with wrong selected position.

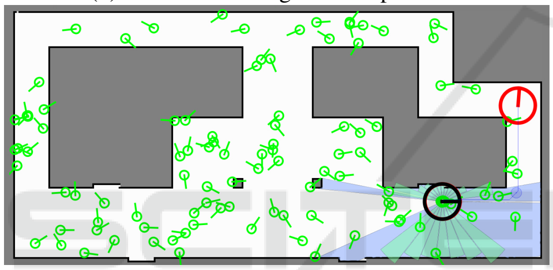

(c) Random KLD with correct position.

Figure 1: Grid Map of changing particle component.

2 RUNNING EXAMPLE

As our running example a driving robot is considered, which can be used in small and large rooms as well as in corridors and outside. For an efficient and accurate calculation, it is necessary to adapt the SLAM algorithm for area changes. Such adjustments can also be made on the basis of a low battery level, in which case an energy efficient algorithm must be selected. Furthermore, requirements for a SLAM algorithm can vary, e.g., special areas are only traversed and in other areas localization must take place with millimeter precision. For all such situations there are specialized SLAM implementations and adaptations, which currently have to be determined manually and implemented in a fixed algorithm.

Our example shows the adjustment of a component of the particle filter and the corresponding particle number at runtime for a Core SLAM (Steux and Hamzaoui, 2010), that uses particle based algorithms for localization in a grid map with a 2D laser scanner. In Figure 1, the process of localization with approaching a target is visualized for a grid map. At the beginning in Figure 1(a), the particle filter KLD (Kullback-Leibler distance) with a few particles is used to save energy. The robot is actually at the bottom left at the beginning. However, the KLD algorithm estimates its position wrongly at the bottom right. This is an immense deviation, which is due to the homogeneous map and the used algorithm. In addition, errors in the robot sensors quickly lead to deviations from the exact position. Throughout the example, the target position is selected at the top right, with a route to be driven drawn as a line from the localized point to the target. Particles are shown in green, with most of them already gathered around the estimated position. In the second step in Figure 1(b), all particles are gathered around the localized position, thus the robot is no longer able to find its correct position.

To remedy this state, the particle component has to be replaced at runtime without affecting other components or the complete SLAM implementation, which would lead to a considerable overhead. In Figure 1(c), this is done by using the Random KLD particle filter, which randomly distributes particles in space and then selects the best one. This algorithm is more computation-intensive but ensures that the localization takes up the correct position of the robot again and that it can still reach its goal.

At this point, it would be possible to switch back to the KLD particle filter after a certain time to save energy and to switch to the Random KLD particle filter if the position probabilities are too low to perform a repositioning. If this approach is no longer applicable due to a low battery level, an adjustment of the maximum number of particles to be reviewed could achieve further energy savings. This example shows the applicability of a product family of SLAM algorithms discussed in the next section and the need of feedback loops for runtime modification.

3 A UNIFYING FRAMEWORK FOR SLAM ALGORITHMS

In the scope of our investigations, we examined 42 SLAM algorithms, comprising both grid- and feature-based methods, listed in Table 1 along with their essential properties. However, only grid-based SLAM methods are incorporated in the feature model in Figure 2. The integration of feature-based methods will be investigated in the future.

To structure Table 1, all SLAM algorithms were examined for their different properties, for which each algorithm is implemented. *Setting* is the basic pro-

Table 1: Features of SLAM algorithms.

Algorithm	Map	Sensors	Feature Detector	Setting	Language
6D SLAM (Quigley et al., 2009)	octo map	2D-laser, D-camera	none	outdoor	C++
Bearing-only SLAM (Kwok and Dissanayake, 2003)	feature map	camera	external	indoor	n/a
CEKF-SLAM (Wang et al., 2012)	feature map	camera	external	indoor, outdoor	Matlab
Core-SLAM or Tiny-SLAM (Steux and Hamzaoui, 2010)	grid map	laser	none	indoor	C++
D-SLAM (Wang et al., 2007)	information matrix	laser	external	indoor	Mathlab
Dense Visual SLAM (Kerl et al., 2013)	pose-sensor graph	RGBD-camera	external	indoor	C++
DP-SLAM (Eliazar and Parr, 2003)	distributed particle grid map	laser	none	indoor	C++
EKF-SLAM (Thrun et al., 2006)	feature map	laser, camera	external	indoor, outdoor	any
EKFmono SLAM (Grasa et al., 2011)	feature map	camera	external	outdoor	Matlab
ESEIF-SLAM (Walter et al., 2007)	graph map, information matrix	any	external	indoor, outdoor, underwater	n/a
Fast Incremental Square Root Information Smoothing (Kaess et al., 2007)	feature map, pose graph	any	external	outdoor	n/a
FastSLAM 1.0 (Thrun et al., 2006)	feature map, grid map	laser, camera	external, none	indoor, outdoor	any
FastSLAM 2.0 (Thrun et al., 2006)	feature map, grid map	laser, camera	external, none	indoor, outdoor	any
FootSLAM (Bruno and Robertson, 2011)	feature map	pedometer	none	walkable	n/a
G^2o-SLAM (Kümmerle et al., 2011)	graph map	any	external	indoor, outdoor	C++
Gmapping (Quigley et al., 2009)	grid map	laser	none	indoor	C++
Graph-SLAM (Thrun et al., 2006)	graph map	laser, camera	external	indoor, outdoor	any
Grid-SLAM (Hahnel et al., 2003)	grid map	laser	none	indoor	C++
Hector SLAM (Kohlbrecher et al., 2014)	grid map	RGBD-camera	optional, none	indoor	C++
HOG-Man SLAM (Grisetti et al., 2010)	graph map	any	external	indoor, outdoor	C++
ICP-SLAM (Tiar et al., 2013)	grid map, pose-sensor graph	laser	none	indoor	C++
iSAM SLAM (Kaess et al., 2008)	feature map, pose graph	laser	external	outdoor	C++, OCaml
iSAM2 SLAM (Kaess et al., 2011)	feature map, graph map	laser	external	indoor, outdoor	n/a
Java FastSLAM (Oursland, 2014)	feature map	any	external	outdoor	Java
Loopy SLAM (Ranganathan et al., 2007)	feature map, pose graph	any	external	outdoor	n/a
Linear SLAM (Zhao et al., 2013)	feature map, pose graph	any	external	indoor, outdoor	Matlab, C++

Table 1: Features of SLAM algorithms. (continued).

Algorithm	Map	Sensors	Feature Detector	Setting	Language
Octo SLAM (Fossel et al., 2013)	octo map	2D-laser	none	indoor, outdoor	C++
Online 6D SLAM (Endres et al., 2012)	octo map	D-camera	none	indoor	C++
PlaceSLAM (Bruno and Robertson, 2011)	feature map	human input	none	walkable	n/a
Range-Only SLAM (Blanco et al., 2008)	feature map, grid map	laser	none	indoor, outdoor, underwater	C++
RatSLAM (Ball et al., 2013)	topological map	camera	external	indoor, outdoor	C++
RBPF-SLAM (Schroeter and Gross, 2008)	grid map	laser	none	indoor, outdoor	C++
RT-SLAM (Roussillon et al., 2011)	feature map	camera	external	indoor	C++
SEIF-SLAM (Thrun et al., 2006)	graph map, information matrix	laser, camera	external	indoor, outdoor	any
SeqSLAM (Milford and Wyeth, 2012)	image graph	camera	external	outdoor	Matlab
SLAM 6D (Nüchter, 2008)	octo map	D-camera	none	indoor	C++
SLAM++ (Salas-Moreno et al., 2013)	feature map, sparse matrix	RGBD-camera	external	indoor	C++
Square Root SLAM (Kaess and Dellaert, 2009)	feature map, pose graph	camera	external	indoor	n/a
SPA-SLAM (Konolige et al., 2010)	sparse matrix	laser	none	indoor	C++
Trajectory-oriented EKF SLAM (Gérossier et al., 2009)	pose-sensor graph	radar sensor	none	outdoor, underwater	Matlab, C++
Vector Field SLAM (Gutmann et al., 2012)	feature map, sparse matrix	infrared, wifi	none	indoor	n/a
WiSLAM (Bruno and Robertson, 2011)	feature map	wifi, pedometer	none	walkable	n/a

perty for which a SLAM process has been designed. A distinction can be made between *indoor*, *outdoor* or *under water*, and *in the air*. However, the difference between individual rooms, long corridors, or a combination of both is also important, as this already limits the use of different SLAM procedures. For the acquisition of environmental data, various *sensors* are needed, ranging from simple *RGB cameras* over *laser scanners* up to *RGBD cameras* (RGB with depth information). The storage options of sensor data are described in the *maps* column and comprises various kinds of maps including grid maps, feature maps and particle-based maps, as show in Table 1.

Grid-based maps are a two-dimensional representation, where a map is divided into square fields each with the status free, occupied, or unknown. An *octo map* adds the third dimension to a grid map retaining all other properties. On the contrary, *feature-based maps* contain identified features and their spacial coordinates to derive the position of the robot based on those features and their position. For *particle-based maps*, particles are assumed to be the estimated positions of the robot, which are randomly derived from the estimated movements to correct hardware errors. Therefore, each computation for a particle can work on its own map or use a shared map. Furthermore, *graph maps* offer the option to store distances between robot positions and features, as well as between both, different features and different robot positions. Further, visibilities and paths are recorded. All other map types are special cases of the presented base types. Those base types differ w.r.t. memory requirements, dimension, and stored information. The *feature detector* column indicates whether features are required to be detected by an external algorithm. As a last criterion, we list the *programming languages* used for the sample implementations, if applicable.

In Figure 2, you can see the feature model for grid-based SLAM methods, which is divided into optional and mandatory elements and can be used to construct new SLAM algorithms. The Core SLAM (Steux and Hamzaoui, 2010) is highlighted in red and represents a valid feature configuration. Core SLAM is a further development of the Fast SLAM (Thrun et al., 2006) algorithm and aims for mapping individual rooms. It is a particle-based process and uses one map for all particles, whereby several particles are determined for one position and the likeliest one is used to update the map. Thereby, the position of the robot is found. As already shown in Section 2, this method can completely lose its orientation in homogeneous rooms. Core SLAM is an example for the adaptation to changing hardware components, because feature recognition is not needed anymore contrary to Fast SLAM, but only 2D laser scanner data.

For reasons of clarity, the feature model does not show dependencies between existing features. For example, the corresponding motion data of the robot is required for updating motion information. In addition, feature recognition algorithms require camera images to work correctly. This example shows the dependencies between sensors, their data formats, and the corresponding algorithms. The described dependencies are considered within our implementation.

Each SLAM algorithm requires *Sensors* (e.g., *Laser Scanners*, *Cameras*, *Depth Image Cameras*, or *RGBD-Cameras*) to identify the environment. These generate *Sensor Data*, which is stored in different map formats. The identified map formats include *Simple Maps* such as graph structures, raster images, or point clouds, as well as *DP maps* and *Particle Lists*, both of which are special *Particle Filter Maps*. The *Working Mode* is critical to the scope of the process, as *Full-SLAM* covers both localization and mapping, whereas *Localization* does only the former. However, if an inaccurate SLAM algorithm is used, the SLAM can distort the maps and render them unusable.

Path Planning does not fit directly into the configuration of a SLAM procedure but is required in most cases and is therefore integrated in the feature model with the *A-Star* and *Dijkstra* algorithms. All other available features are optional and assemble the SLAM algorithm.

The feature *Motion Data* distinguishes between different motion models of a robot, where *Velocity Data* defines a forward motion with a circular motion, and *Odometry Data* a sequential execution of rotary and motion movements. The feature *EKF* (Extended Kalman Filter) is used to draw conclusions about the real state caused by faulty measurement and movement data, based on information about the resilience of the hardware. Algorithms for *Feature Generation* are only used for camera recordings and search for landmarks in them.

A large part of the feature model includes particle-based features, which are divided into *Sample Motion*, *Resampling*, *Clusterer*, and *Particle Number*. All mentioned features except cluster algorithms have already been presented in Section 2. Cluster algorithms are used to close loops within mapping environments. These algorithms create a closed map and remove incorrect overlaps of information.

Algorithms for direct determination of the robot position based on motion and sensor data are summarized in the feature *Position Estimate* including specifications of *ICP*, *IDC*, and *IMRP* for 2D and 3D maps.

The last branch of the model includes probability models, which determine a probability value for the current position based on the robot position and sensor data. In particular, map information and laser lines of the scanner are included for this purpose. The current feature model contains only raster-based SLAM methods. However, it offers the possibility to configure and create SLAM procedures based on self-selected properties.

4 RUNTIME RECONFIGURATION OF SLAM ALGORITHMS

This section introduces a concept for reconfiguration of a SLAM algorithm at runtime, using the feature model described in Section 3 as a basis for creating new variants. Figure 3(a) shows the basic process of static variability modeling, which is taken as the basic building block for our approach, wherein a new executable is created from a variant of the feature tree. To do this, a solution model must exist for a feature model that maps all features to elements in this model. In the mapping phase, a modified model is created for the selected variant, from which a configuration and code is generated. This generated code can be compiled and started as an executable program and replaces the current running model. The process is sufficient for static reconfiguration but does not include any information from the running model and creates overhead as the complete program is exited and restarted. This also means that the collected data must be saved and reloaded. To improve efficiency, we extend the process of static variability modeling by integrating the MAPE-K feedback loop. MAPE-K contains the phases Monitor, Analyze, Plan, and Execute. In addition, the MAPE-K loop contains a knowledge base that collects data from the running system. With it, the feedback loop permanently observes the underlying system and reacts on changes.

Figure 3(b) shows the complete concept for runtime reconfiguration of SLAM algorithms. In the monitor phase, the sensor and robot data are recorded and

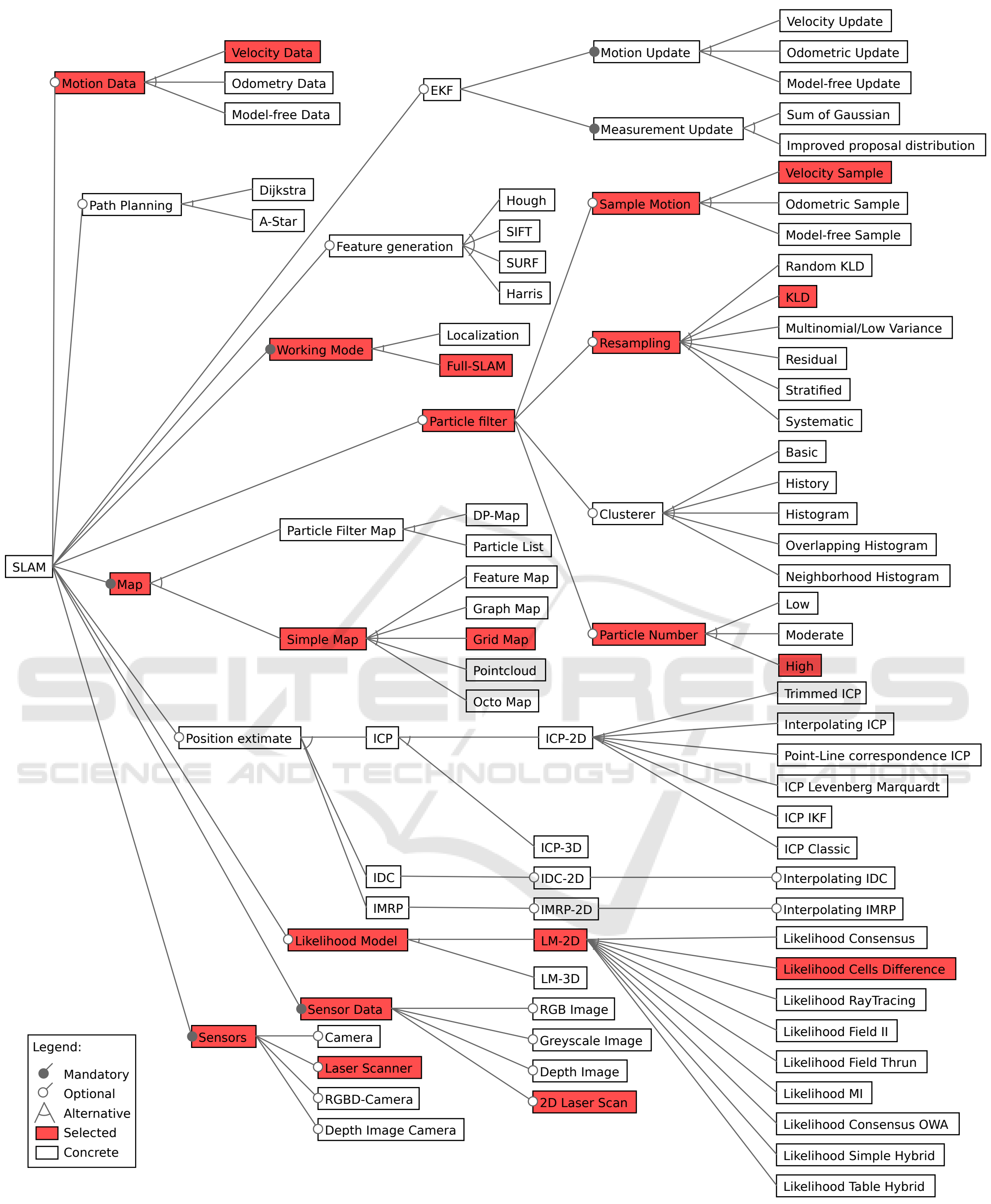
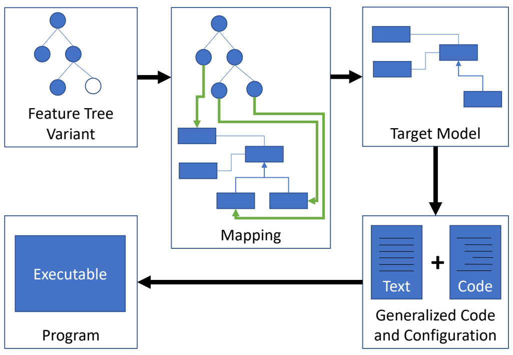
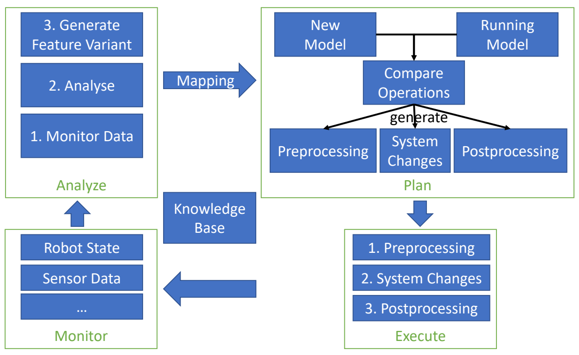


Figure 2: The Feature Model with Core SLAM as selected feature configuration.

passed on to the analysis phase. Therein, the data is analyzed using various parameters and used to check the efficiency of the currently running SLAM configuration. If the analysis finds optimization possibilities, a new feature variant is generated. Otherwise, the running program is not changed. In the case of optimization, the new feature configuration is mapped to a new model and transferred to the planning phase. This compares the current model with the new model to determine changes. Based on the identified



(a) Static variablility.



(b) MAPE-K with dynamic runtime reconfiguration.

Figure 3: Concept for Reconfiguration.

modifications, preprocessing steps, system changes, and postprocessing steps are determined. As a basic rule, the earlier a change occurs in the feature model, the more process steps are required for the adaptation. Adapting a particle based on a position estimate SLAM requires complete parts of the program to be exchanged, which leads to interruptions in the processing. However, switching between different resampling algorithms can be performed within one step without creating an interruption. In the static variability modeling, a new executable would have to be created and started for each new configuration. As soon as the planning phase has determined the process sequence, it is transferred to the execute phase. This phase manages the interruptions for the robot and executes the reconfiguration steps in the predefined order, e.g., in case multiple algorithms have to be instantiated. Therefore, the feedback loop can modify the running system based on the monitored runtime data collected continuously. In the next section, the implementation of this concept is shown, with the focus on the plan and execute phase, since the SLAM procedures were examined only for their features. Neither accuracy nor efficiency were considered yet, thus no analysis algorithm can be derived.

5 EVALUATION BY IMPLEMENTATION

In this section, the steps to get from one instance of the feature model to a working, adaptable implementation are detailed.

We base our work on the component-based framework GeneralRobot developed at the HTW Dresden. Its main concepts are processors and data containers. *Processors* are responsible for a certain computation and running constantly triggered periodically or upon new data is available. On the other hand, *data containers* only hold data, are read by processors and inform them about changes. To configure the used processors and data containers, GeneralRobot uses a simple configuration file. We use a properties model describing key-value pairs to define used components and their configuration parameter values.

Getting from a feature configuration to such a properties model needs a mapping. We use the *FeatureMapper* (Heidenreich et al., 2008) to map each feature to a set of key-value-pairs. It uses the 150% approach (Weißleder and Lackner, 2013), i.e., all components in the target model are already there and will be removed based on the feature selection. Using the FeatureMapper, we can automatically generate a properties model from a given feature configuration. The feature configuration is depicted in Figure 4.

To use this properties model, we made slight modifications to the framework GeneralRobot, e.g., add methods to update used components or configuration values, or to modify processors to change subprocessors in a thread safe way. Generating a new properties model after a change in the feature model, the framework compares the current properties with the new ones and derive reconfiguration steps. Those steps can be small, like changing a configuration parameter, or bigger adaptations such as exchanging a set of processors. All changes will be reordered to avoid unnecessary work, e.g., changing a parameter before replacing the component using this parameter.

Coming back to the example described in Section 2, in the following we describe what happens to change from KLD to Random KLD. The base feature configuration is depicted in Figure 2 where KLD is selected. From this, a properties model was created and loaded by GeneralRobot. To start reconfiguration, the feature *KLD* deselected and *Random KLD* is selected. Then, the FeatureMapper is run again to generate a new properties model, which now includes a property using the class `RandomKldSampling` for resampling. This model is picked up inside our mo-

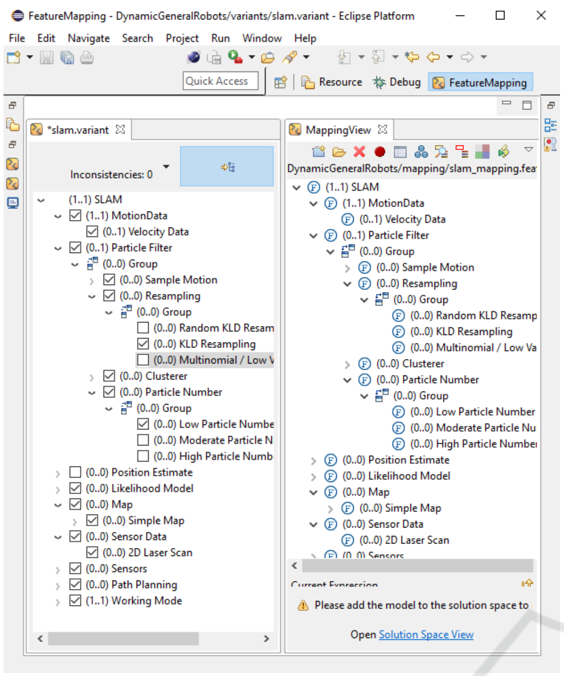


Figure 4: Screenshot of the FeatureMapper view.

dified version of GeneralRobot, where it is compared to the current properties model to derive the needed reconfiguration steps. In this case, the object of type `KldSampling` is disabled, i.e., it is stopped and connections to data containers are removed. Afterwards a new object of type `RandomKldSampling` is created and wired to the same containers. The important point here is, that because of the separation of concerns followed by GeneralRobot, only the processors need to be replaced and the data containers are left untouched and intermediate data is preserved. The described process eliminates the need for shut down and restart of the running system altogether. Thus, it increases the adaptability and performance of the overall system and with it the reuse of common SLAM functionality.

6 RELATED WORK

In the research area for SLAM, individual algorithms are usually created, which are optimized for a certain type of environment or in a partial area of the algorithm, but not frameworks, which combine the advantages of different SLAM methods. In this section, we want to present four robot frameworks, which offer several SLAM methods and prescribe fixed interfaces for the communication of the subcomponents. Section 5, our extension to the GeneralRobot Framework from the HTW Dresden is already presented, using our feature model to transfer from the static basic programming into a dynamic framework for SLAM algorithms.

The Robot Operating System (ROS) (Quigley et al., 2009) platform is an open source project, which deals especially with robot environments. It provides a complete framework with interfaces and implementations for sensors, robot controllers, localization, mapping, and control. Two SLAM methods, GMapping (Grisettiyz et al., 2005) and HectorSLAM (Kohlbrecher et al., 2014) are implemented in ROS. These are integrated into the ROS platform as complete modules and not as partial algorithms and can therefore only be exchanged as a whole. This makes it difficult to modify and adapt a SLAM algorithm. For ROS, however, the SLAM algorithms only represent an extension of the platform. In addition, it is possible to integrate frameworks such as OpenCV for processing sensor data such as camera images or laser scanner data but this does not facilitate the development and integration of further SLAM methods.

The OpenSLAM (Stachniss et al., 2007) platform collects a variety of SLAM algorithms and implementations and makes them available to researchers and users. Users can publish their SLAM implementations on an internet platform with a small documentation of the system requirements.These must be stable but are not linked in any framework or software product. Due to the large selection of SLAM methods, the platform offers a good opportunity to analyze the individual algorithms and the structure of the SLAMs. However, the granularity of the implementation of SLAM implementations plays a major role here. This ranges from complete implementation in one method containing the complete communication and function algorithms to splitting into several methods. The results are multiple implementations of different algorithms. Therefore, OpenSLAM provides a good basis for extending the existing software.

The Mobile Robot Programming Toolkit (Claraco, 2008) platform goes one step further than OpenSLAM (Stachniss et al., 2007) and offers a complete framework programmed in C/C++. The framework prescribes data exchange formats and data structures to which developers must orient themselves without any help from the toolkit to create a new adapted algorithm. The framework consists of three parts: (1) movement planning with different navigation and control algorithms, (2) computer vision for the analysis of sensor data, and (3) SLAM methods and algorithms. It currently includes Graph, EKF, Fast, RBPF, and ICP-SLAM implementations but provides those without usage rules, which transfers a lot of usage effort to programmers. Furthermore, this platform offers implementations for communication with hardware components such as sensors and robots. With the functionality and structure of the framework, it offers a good extension point for the current feature tree and the current implementation.

7 CONCLUSION

In this paper, in Section 3, we presented a novel feature-based classification of SLAM algorithms as a result of analyzing more than 40 existing algorithms as shown in Table 1 and Figure 2. Next, in Section 4, we introduced an approach to switch between SLAM algorithms at runtime based on this classification. We evaluated our approach using a prototypical implementation as described in Section 5.

Using the proposed classification reusable and exchangeable parts of SLAM algorithms are identified. In consequence, the variation or exchange of running SLAM algorithms is enabled. This, in turn, allows to always use the most adequate algorithm depending on the current framing conditions (e.g., in- vs. outdoor).

In future, we will extend the feature model and add results from analyzing feature-oriented SLAM algorithms and other new implemented SLAMs. This leads to a higher amount of SLAM variants and creates the needs for extending the implementations of the plan and execute phases to add adaptable analyze algorithms for robots.

ACKNOWLEDGMENTS

This work has been funded by the German Research Foundation within the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing", within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907) and by the German Federal Ministry of Education and Research within the project "OpenLicht".

REFERENCES

Ball, D., Heath, S., Wiles, J., Wyeth, G., Corke, P., and Milford, M. (2013). OpenRatSLAM: an open source brain-based SLAM system. *Autonomous Robots*, 34(3):149–176.

Blanco, J. L., Fernandez-Madrigal, J. A., and Gonzalez, J. (2008). Efficient probabilistic Range-Only SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1017–1022.

Bruno, L. and Robertson, P. (2011). WiSLAM: Improving FootSLAM with WiFi. In *International Conference on Indoor Positioning and Indoor Navigation*, pages 1–10.

Claraco, J. L. B. (2008). Development of scientific applications with the mobile robot programming toolkit. *The MRPT reference book. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Spain*.

Eliazar, A. and Parr, R. (2003). DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *18th International Joint Conference on Artifical Intelligence*, volume 3, pages 1135–1142.

Endres, F., Hess, J., Engelhard, N., Sturm, J., and Burgard, W. (2012). Online- 6D-SLAM für RGB-D-Sensoren. *Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik*, 60(5):270–278.

Fossel, J., Hennes, D., Claes, D., Alers, S., and Tuyls, K. (2013). OctoSLAM: A 3D mapping approach to situational awareness of unmanned aerial vehicles. In *International Conference on Unmanned Aircraft Systems*, pages 179–188.

Grasa, O. G., Civera, J., and Montiel, J. M. M. (2011). EKF monocular SLAM with relocalization for laparoscopic sequences. In *IEEE International Conference on Robotics and Automation*, pages 4816–4821.

Grisetti, G., Kümmerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010). Hierarchical optimization on manifolds for online 2D and 3D mapping. In *IEEE International Conference on Robotics and Automation*, pages 273–278.

Grisettiyz, G., Stachniss, C., and Burgard, W. (2005). Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *IEEE International Conference on Robotics and Automation*, pages 2432–2437.

Gérossier, F., Checchin, P., Blanc, C., Chapuis, R., and Trassoudaine, L. (2009). Trajectory-oriented EKF-SLAM using the Fourier-Mellin Transform applied to Microwave Radar Images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4925–4930.

Gutmann, J. S., Eade, E., Fong, P., and Munich, M. E. (2012). Vector Field SLAM–Localization by Learning the Spatial Variation of Continuous Signals. *IEEE Transactions on Robotics*, 28(3):650–667.

Hahnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Heidenreich, F., Kopcsek, J., and Wende, C. (2008). FeatureMapper: Mapping Features to Models. In *Companion of the 30th International Conference on Software Engineering*, pages 943–944. ACM.

Kaess, M. and Dellaert, F. (2009). Covariance recovery from a square root information matrix for data association. *Robotics and autonomous systems*, 57(12):1198–1210.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2011). isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE International Conference on Robotics and Automation*, pages 3281–3288.

Kaess, M., Ranganathan, A., and Dellaert, F. (2007). Fast Incremental Square Root Information Smoothing. In *20th International Joint Conference on Artifical Intelligence*, pages 2129–2134. Morgan Kaufmann Publishers Inc.

Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378.

Kerl, C., Sturm, J., and Cremers, D. (2013). Dense visual SLAM for RGB-D cameras. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). G^2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, pages 3607–3613.

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., and von Stryk, O. (2014). Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots. In *RoboCup 2013: Robot World Cup XVII*, pages 624–631. Springer.

Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., and Vincent, R. (2010). Efficient Sparse Pose Adjustment for 2D mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 22–29.

Kwok, N. and Dissanayake, G. (2003). Bearing-only slam in indoor environments using a modified particle filter. In *Australasian Conference on Robotics and Automation*. University of Queensland.

Milford, M. J. and Wyeth, G. F. (2012). SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. In *IEEE International Conference on Robotics and Automation*, pages 1643–1649.

Nüchter, A. (2008). *3D robotic mapping: the simultaneous localization and mapping problem with six degrees of freedom*, volume 52. Springer.

Oursland, A. (2014). Java FastSLAM. *http://www.oursland.net/projects/fastslam/*.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5.

Ranganathan, A., Kaess, M., and Dellaert, F. (2007). Loopy SAM. In *20th International Joint Conference on Artifical Intelligence*, pages 2191–2196. Morgan Kaufmann Publishers Inc.

Roussillon, C., Gonzalez, A., Solà, J., Codol, J.-M., Mansard, N., Lacroix, S., and Devy, M. (2011). RT-SLAM: A Generic and Real-Time Visual SLAM Implementation. In *Computer Vision Systems*, pages 31–40. Springer.

Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H., and Davison, A. J. (2013). Slam++: Simultaneous localisation and mapping at the level of objects. In *Computer Vision and Pattern Recognition*, pages 1352–1359. IEEE.

Schroeter, C. and Gross, H. M. (2008). A sensor-independent approach to RBPF SLAM - Map Match SLAM applied to visual mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2078–2083.

Stachniss, C., Frese, U., and Grisetti, G. (2007). OpenSLAM. *http://www.openslam.org*.

Steux, B. and Hamzaoui, O. E. (2010). tinySLAM: A SLAM algorithm in less than 200 lines C-language program. In *11th International Conference on Control Automation Robotics Vision*, pages 1975–1979.

Thrun, S., Fox, D., and Burgard, W. (2006). *Probabilistic Robotics*. The MIT Press.

Tiar, R., Ouadah, N., Azouaoui, O., Djehaich, M., Ziane, H., and Achour, N. (2013). ICP-SLAM methods implementation on a bi-steerable mobile robot. In *IEEE 11th International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics*, pages 1–6.

Walter, M. R., Eustice, R. M., and Leonard, J. J. (2007). Exactly Sparse Extended Information Filters for Feature-Based SLAM. *The International Journal of Robotics Research*, 26(4):335–359.

Wang, H., Li, C., Lv, H., and Chen, X. (2012). Research on compressed EKF based SLAM algorithm for unmanned underwater vehicle. In *15th International IEEE Conference on Intelligent Transportation Systems*, pages 1402–1406.

Wang, Z., Huang, S., and Dissanayake, G. (2007). D-SLAM: Decoupled Localization and Mapping for Autonomous Robots. In Thrun, S., Brooks, R., and Durrant-Whyte, H., editors, *Robotics Research*, pages 203–213. Springer.

Weißleder, S. and Lackner, H. (2013). Zwei Ansätze zur automatischen modellbasierten Generierung von Testfällen für variantenreiche Systeme. *Softwaretechnik-Trends*, 33(2).

Zhao, L., Huang, S., and Dissanayake, G. (2013). Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 24–30.